Computational Assignment Report

Manakovskiy Daniil
BS18-02
Innopolis University

09/11/2019

0 Note

The source code [3] and all images from the report [2] are available at the GitHub repository.

1 The task

Variant 14.
Given equation:

$$y' = \left(1 + \frac{y}{x}\right) \ln\left(\frac{x+y}{y}\right) + \frac{y}{x},\ \frac{y}{x} > -1,\ , x \neq 0 \tag{1}$$

Given IVP:

$$\begin{cases} y_0 = 2 \\ x_0 = 1 \end{cases} \tag{2}$$

Given end of Interval:

$$X = 6 \tag{3}$$

2 Exact solution

Equation 1 can be rewritten as:

$$y' = \left(1 + \frac{y}{x}\right) \ln\left(1 + \frac{y}{x}\right) + 1 + \frac{y}{x} - 1$$

Let:

$$z = 1 + \frac{y}{x},\ z > 0$$

Then:

$$\begin{aligned} y &= xz - x = x(z-1) \\ y' &= z - 1 + xz' \end{aligned}$$

The equation 1 can be transformed to:

$$\begin{aligned} z - 1 + xz' &= z\ln z + z - 1 \\ z'x &= z\ln z \end{aligned}$$

The obtained equation is separable:

$$\begin{aligned}
\frac{\mathrm{d}z}{\mathrm{d}x}x &= z\ln z \\
\frac{\mathrm{d}z}{z\ln z} &= \frac{\mathrm{d}x}{x} \\
\int\frac{\mathrm{d}z}{z\ln z} &= \int\frac{\mathrm{d}x}{x} \\
\int\frac{\mathrm{d}\ln z}{\ln z} &= \int\frac{\mathrm{d}x}{x} \\
\ln|\ln z| &= \ln|x| + C,\ C\in\mathbb{R} \\
\ln z &= Cx \\
z &= e^{Cx}
\end{aligned}$$

Substituting back we obtain:

$$y = x\left(e^{Cx}-1\right), x\neq 0, y > -x, C\in\mathbb{R}$$

Find C:

$$\begin{aligned}
e^{Cx_0} &= \frac{y_0}{x_0}+1 \\
C &= \frac{1}{x_0}\ln\left(\frac{y_0}{x_0}+1\right) \\
x_0 &\neq 0, y_0 > -x0
\end{aligned}$$

For the given IVP:

$$\begin{aligned}
C &= \ln(3) \\
y &= x(3^x-1) \\
x &\in \mathbb{R}
\end{aligned}$$

An exact solution in general is

$$y = x\left(e^{\frac{x}{x_0}\ln\left(\frac{y_0}{x_0}+1\right)}-1\right), x_0\neq 0, y_0 > -x_0 \tag{4}$$

At the given interval $x\in(x_0,X)=(1,6)$ there is no discontinuity points

3 Application

The outcome of the assignment is a program for calculating numerical solution using different methods (Euler, Improved Euler, Runge-Kutta), analysing them and comparing with an exact solution.

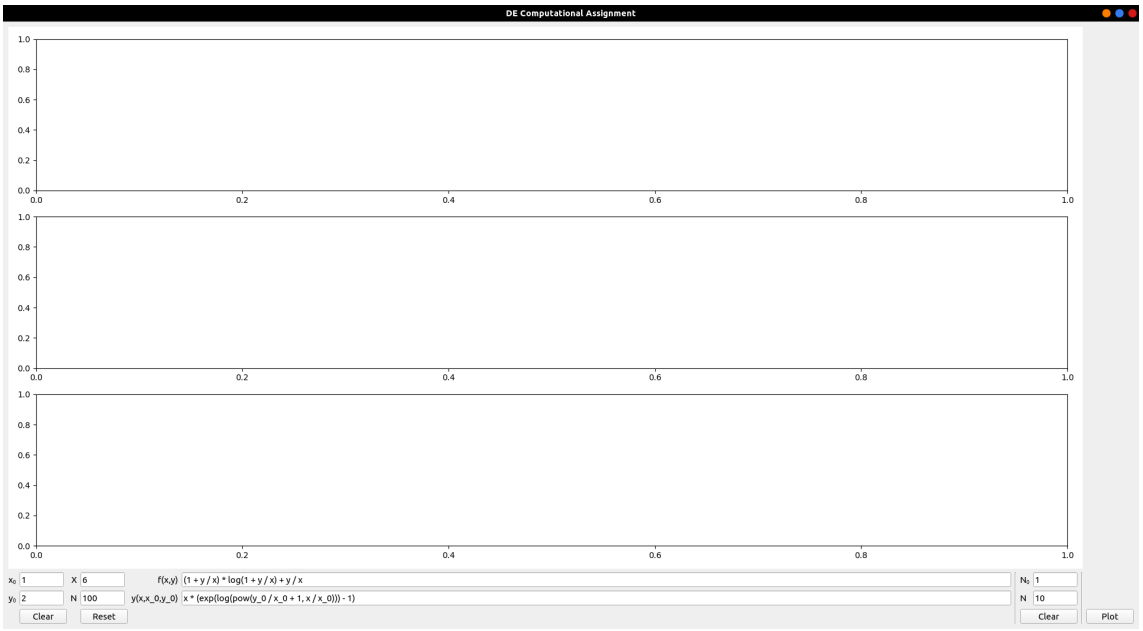

Figure 1: Interface without graphs.

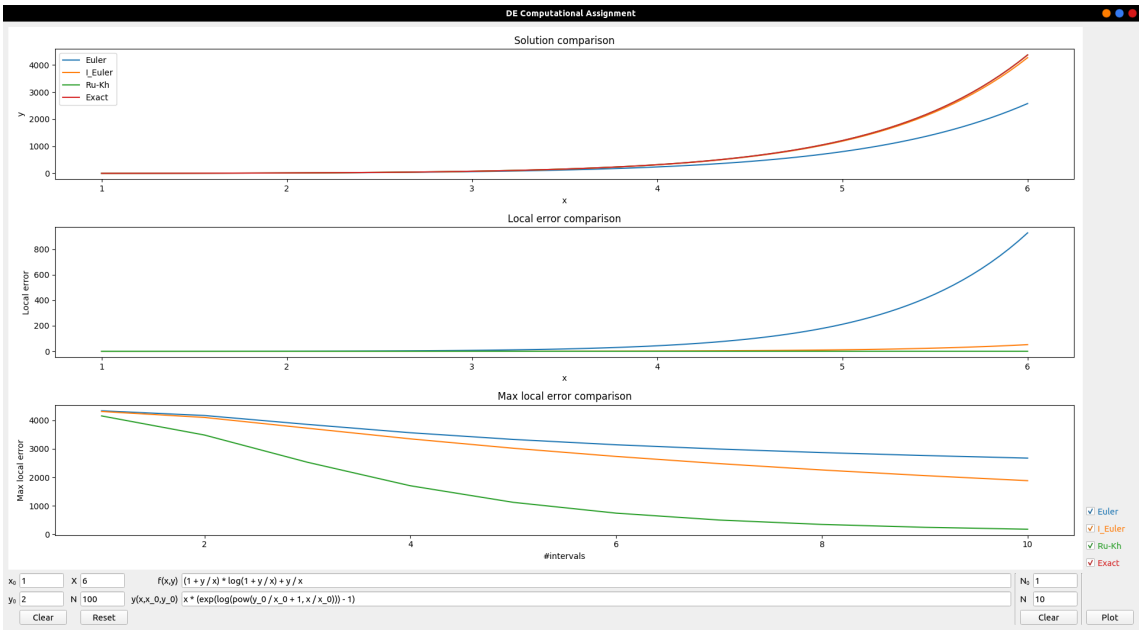

Figure 2: Interface with graphs.

3.1 Requirements

Requirements for building the code from source code [3] are the following:

- Python 3.6
- Matplotlib
- NumPy
- PyQt5

3.2 Features

- One-file executable file independent of environment.
- Possibility to change Initial value problem, clear all fields, reset to default.
- Ability to change initial and exact function (with validation to comply to the IVP).
- Disabling graphs on plot.
- Adaptive grid.

3.3 Input

As it is shown at figure 1, the application allows user to set the following parameters.

Section for solving IVP:

- x_0 and y_0 - Initial Value for a differential equation.
- X - end point for numerical solutions.
- N - number of intervals for numerical solutions.
- $f(x,y)$ - a derivative of the function y.
- $y(x,x_0,y_0)$ - an exact solution for the given IVP with an explicit constant.

Max local error section:

- N_0 - starting number of intervals.
- N - end number of intervals.

3.3.1 Validation

x_0, y_0, X must be dot-separated rational numbers.
N, N_0 must be positive integers.

The following conditions must hold:

- IVP section:
  - $y(x_0,x_0,y_0) = y_0$
  - y and f functions must comply to python math library [1] and refer to initial values as x_0 and y_0.
- Max local error section:
  - $N_0 > N$

If any of conditions above does not satisfied, user will be prompted to fix the input (figure 3).

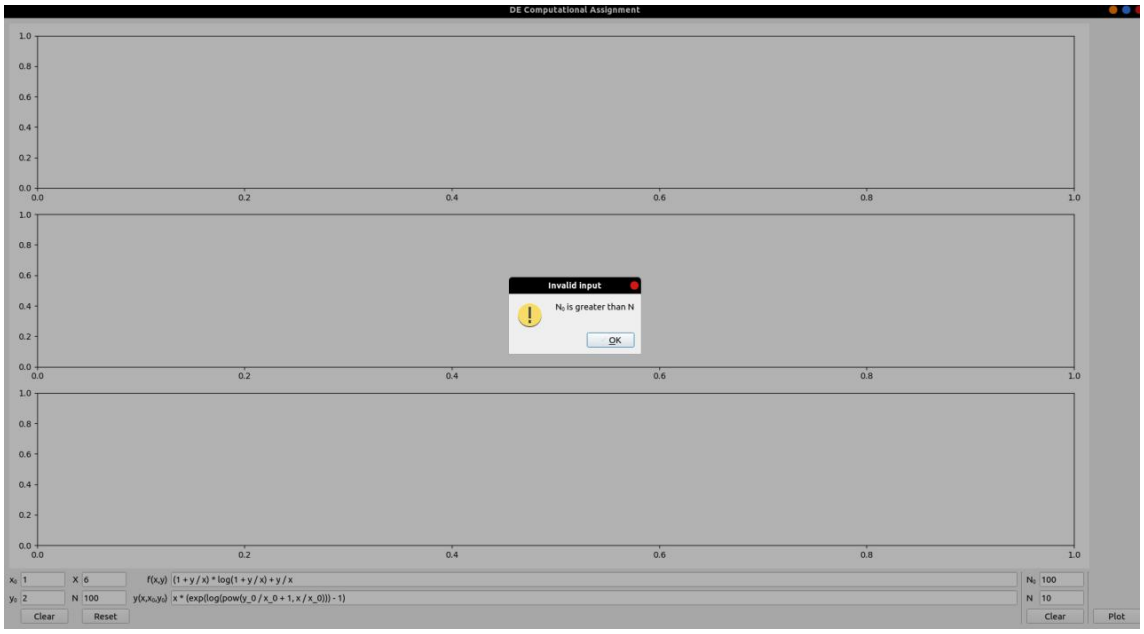

Figure 3: Invalid input message

3.3.2 Default values

As mentioned in section 3.2 there is a reset button. Pressing the button resets all input values to the default. Defaults are the following:

- IVP section:
  - $x_0 = 1$ according to the IVP (2).
  - $y_0 = 2$ according to the IVP (2).
  - $X = 6$ according to the given variant (3).
  - $N = 100$
  - $f(x,y) = (1+y/x)*log(1+y/x)+y/x$ according to the given function (1).
  - $y(x,x_0,y_0) = x*(exp(log(pow(y\_0/x\_0+1,x/x\_0)))-1)$ according to calculated solution (4).
- Max local error section:
  - $N_0 = 1$
  - $N = 10$

3.4 Output

The output of a program is 3 plots that allows comparing performance of different numerical methods (figure 2).

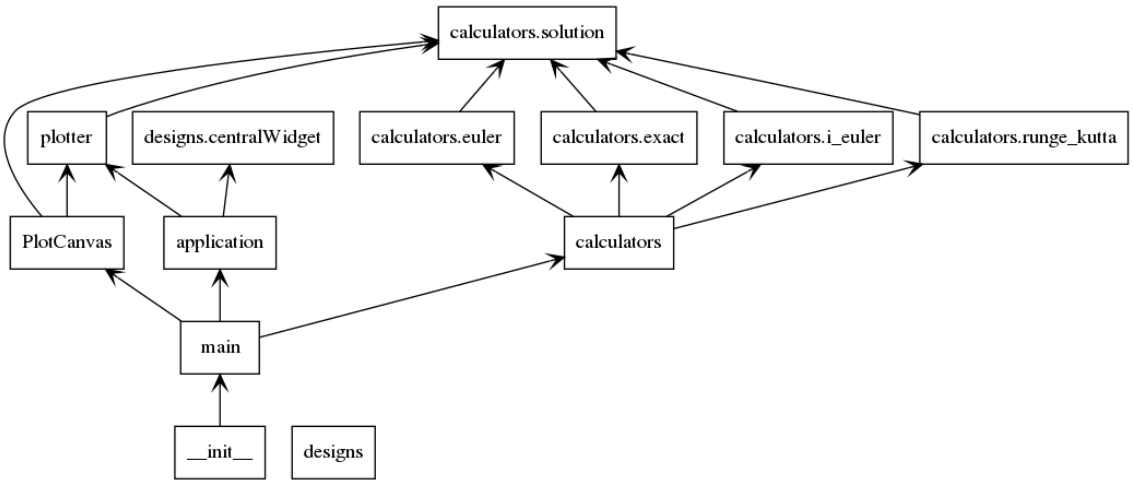

Figure 4: Package diagram

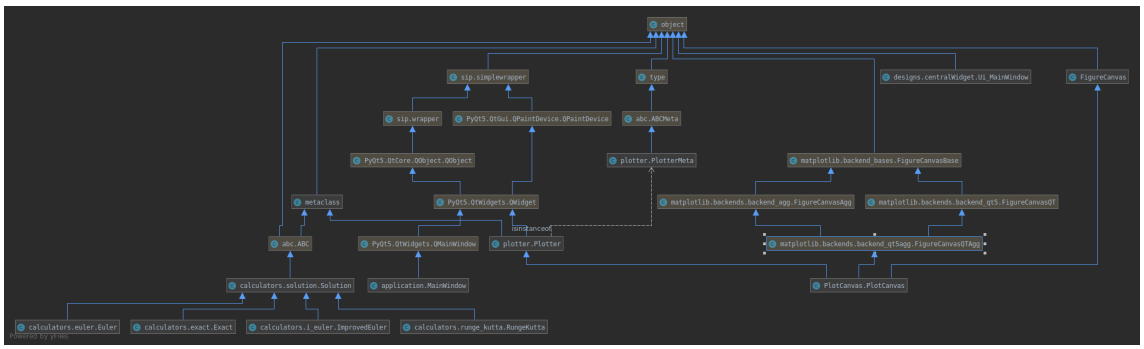

Figure 5: Class diagram

3.5 Program structure

Detailed structure of the application can be seen at package diagram (figure 4), class diagram with internal classes included (figure 5) and expanded class diagram of newly created classes (figure 6). The source code obeys SOLID principle: every part is responsible over single part of functionality, all modules are extendable and interchangeable up to abstraction.

The challenging part was to create an abstraction for a plotter class due to features in implementations of PyQt5 python integration and Python's classes. In theory a plotter is a subclass of PyQt5.QWidget with some desired functions. Since QT originally implemented in C++, all PyQt classes are C++ bindings, not pure python types, thus multiple inheritance from a binding (sip.wrappertype) and a python class results into TypeError exception caused by super-classes' layout conflict at C level. The way to overcome this problem is shown at listings 1 and 2. The trick is to create metaclass PlotterMeta (listing 1) that inherits from type(QWidget)=sip.wrappertype that is a subclass of metaclass type, then create actual abstraction Plotter which is a subclass QWidget with a metaclass PlotterMeta. Thus inheriting PlotCanvas (listing 2) from FigureCanvas (matplotlib integration) and Plotter produces no errors since both super-classes are subclasses of sip.wrappertype and have compatible layout.

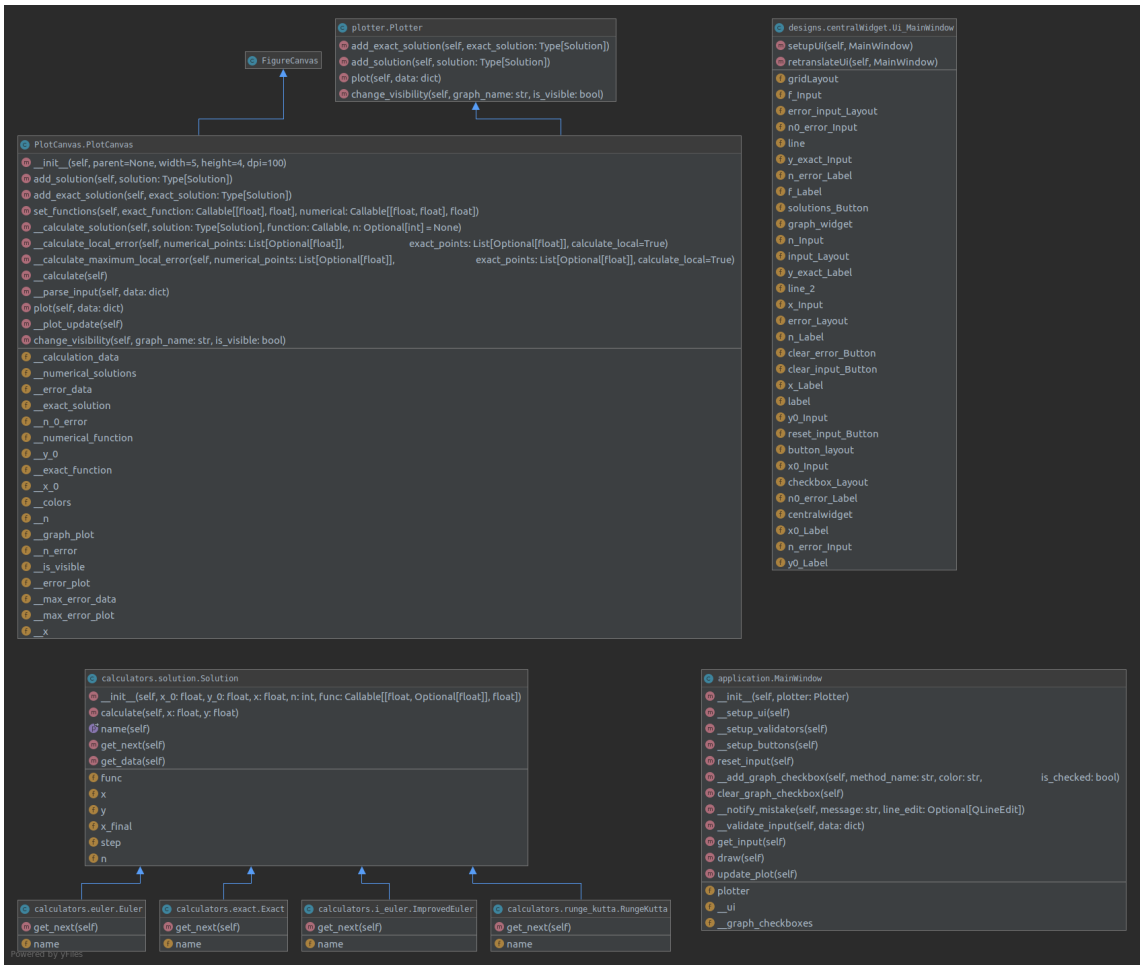

Figure 6: Expanded class diagram of main classes

Source Code 1: Implementation of Plotter ABC

```python
from typing import Type, List, Tuple
from abc import ABCMeta, abstractmethod

from PyQt5.QtWidgets import QWidget
from calculators.solution import Solution


class PlotterMeta(type(QWidget), ABCMeta):
    pass


class Plotter(QWidget, metaclass=PlotterMeta):
    @abstractmethod
    def add_exact_solution(self, exact_solution: Type[Solution]) -> None:
        raise NotImplementedError

    @abstractmethod
```

```
    def add_solution(self, solution: Type[Solution]) -> None:
        raise NotImplementedError

    @abstractmethod
    def plot(self, data: dict) -> List[Tuple[str, Tuple[str, bool]]]:
        # Input format must be specified by a subclass
        raise NotImplementedError

    @abstractmethod
    def change_visibility(self, graph_name: str, is_visible: bool) -> None:
        # Change visibility of a graph by name
        raise NotImplementedError
```

Source Code 2: Implementation of PlotCanvas

```
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure

from calculators.solution import Solution
from plotter import Plotter


class PlotCanvas(FigureCanvas, Plotter):
    ...
```

4 Result Analysis

The result of calculation allow us to compare performance of different methods. Let consider the result at #intervals = 100.

4.1 Euler method

Overall comparison can be seen at figure 2. It is clear that for given function Euler method's approximation has relatively big local error (up to 928.584) starting from $x = 3.5$ as well as global error (1792.719) due to using small amount of parameters to perform approximation.

4.2 Improved Euler

Improved Euler method performs much better then Euler, since it consider average slope of a function, that is definitely better for such exponential function. This method also have dramatic relative increase in local error (up to 52.032), but this time an error is not so big, and global error is just 100.376

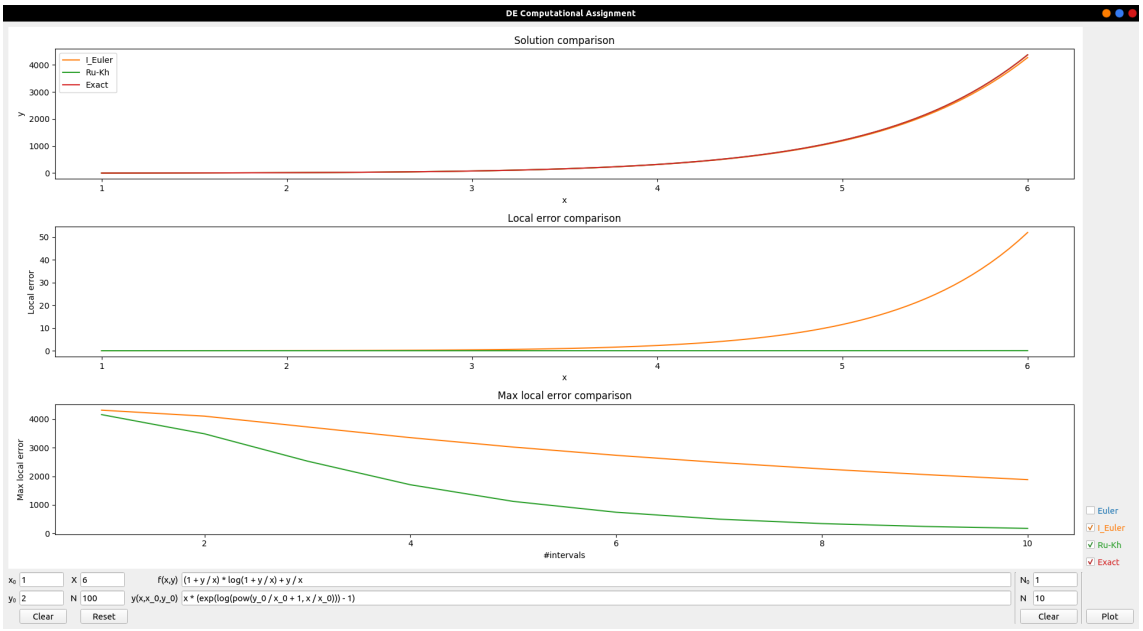

Figure 7: Graphs without Euler method

4.3 Runge-Kutta method

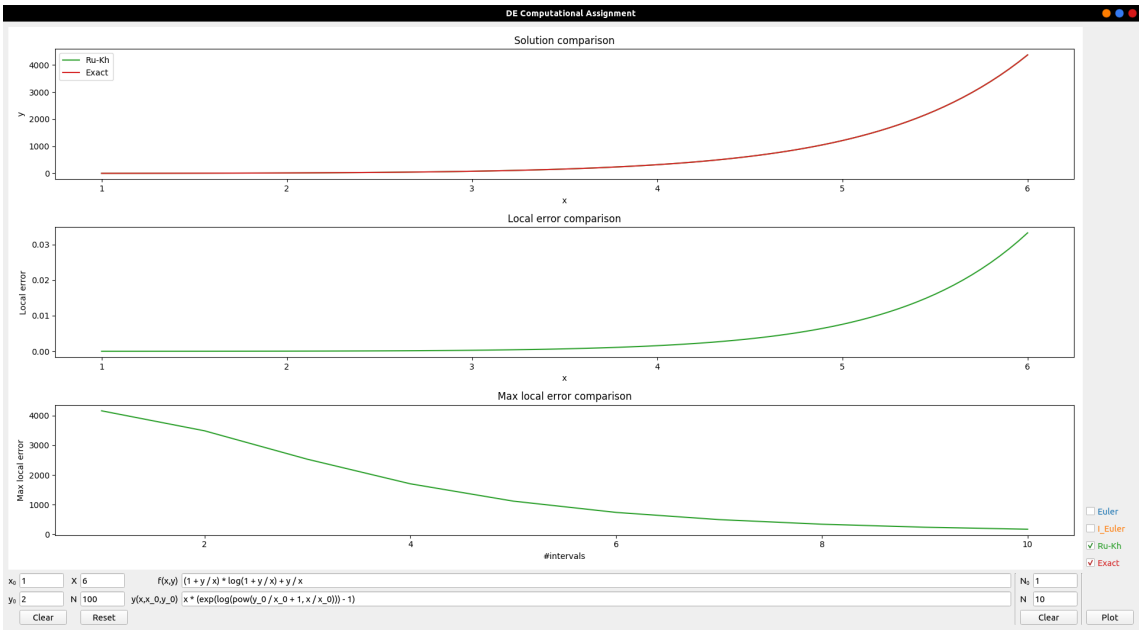

Figure 8: Runge-Kutta method in comparison with the exact solution

The best approximation is obtained using Runge-Kutta method. As it shown at figure 8, this method, as well as others, loses accuracy after $x = 3.5$, but the approximation error is negligible (up to 0.031) with the total error = 0.064

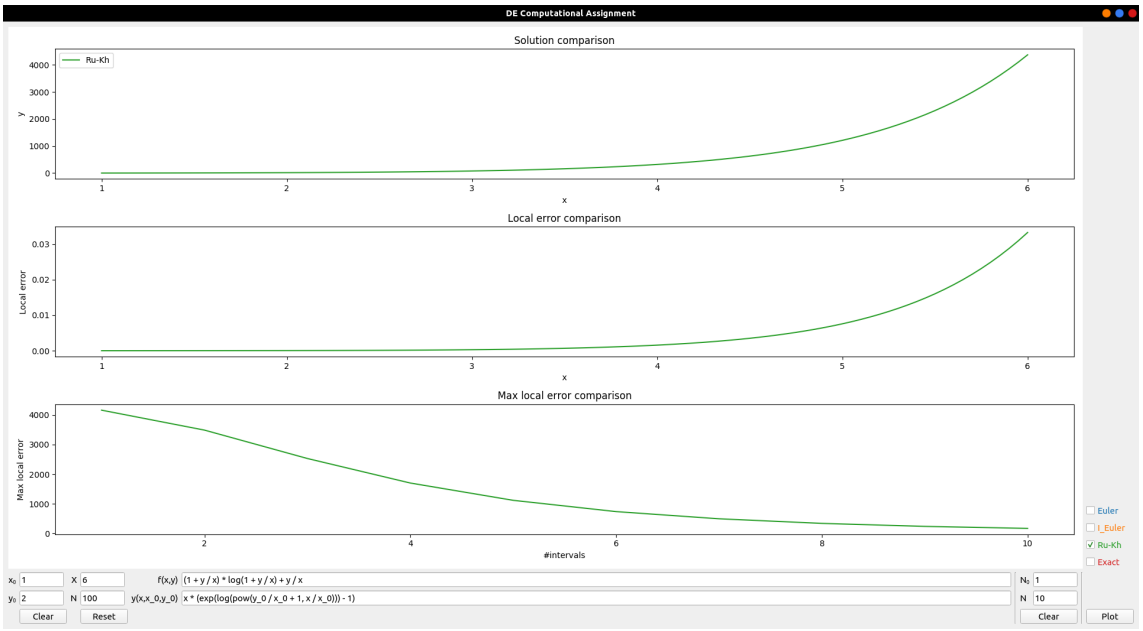

Figure 9: Runge-Kutta method's graph

5 Conclusions

As a result, imperially has been proven that for the given function and continuous functions from different variants Runge-Kutta method performs much better that others achieving relatively small error even on functions that grows exponentially fast.

References

[1] P. S. Foundation. Mathematical functions in python 3.7, 2019. https://docs.python.org/3.7/library/math.html, Last accessed on 09/11/2019.

[2] D. Manakovskiy. Computational assignment report images, 2019. https://github.com/WinnerOK/DE-numerical-methods/tree/master/Report, Last accessed on 09/11/2019.

[3] D. Manakovskiy. Computational assignment source code, 2019. https://github.com/WinnerOK/DE-numerical-methods, Last accessed on 09/11/2019.